Assignment: Comparitive study on Multivariable Linear Regression

Pranay Shukla
Roll Number: 24075069
Email: pranay.shukla.cse24@iitbhu.ac.in

May 31, 2025

Contents

1 Introduction to Multivariable Linear Regression

Multivariable linear regression is a supervised learning algorithm used to model the relationship between multiple input features and a continuous target variable. It extends simple linear regression by incorporating more than one feature, allowing the model to fit a hyperplane to high-dimensional data.

In this project, we evaluate three different implementations:

- Pure Python
- NumPy
- Scikit-learn

Mathematical Formulation

$$f_{\mathbf{w},b}(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} + b \tag{1}$$

$$J(\mathbf{w}, b) = \frac{1}{2m} \sum_{i=0}^{m-1} \left(f_{\mathbf{w},b}(\mathbf{x}^{(i)}) - y^{(i)} \right)^2 \tag{2}$$

Gradient Descent:

$$w_j = w_j - \alpha \cdot \frac{\partial J}{\partial w_j} \quad \text{for } j = 0 \ldots n-1 \tag{3}$$

$$b = b - \alpha \cdot \frac{\partial J}{\partial b} \tag{4}$$

Gradient Expressions:

$$\frac{\partial J}{\partial w_j} = \frac{1}{m} \sum_{i=0}^{m-1} (f_{\mathbf{w},b}(\mathbf{x}^{(i)}) - y^{(i)}) x_j^{(i)} \tag{5}$$

$$\frac{\partial J}{\partial b} = \frac{1}{m} \sum_{i=0}^{m-1} (f_{\mathbf{w},b}(\mathbf{x}^{(i)}) - y^{(i)}) \tag{6}$$

2 Data Preprocessing

2.1 Dataset Overview

We used the California Housing dataset from Kaggle.[1] The dataset contains 20,640 rows and 10 columns like `median_income`, `total_rooms`, `population`, and `ocean_proximity`.

[1] `https://www.kaggle.com/datasets/camnugent/california-housing-prices`

2.2 Cleaning and Transformation

- Removed 207 rows with missing values (NaN) in `total_bedrooms`
- Applied OneHotEncoding to `ocean_proximity` to generate 5 binary columns as it earlier contained textual data like `Inland` and `Near Bay` etc.
- Removed rows with `median_house_value` $\geq$ 500000 (985 rows), due to data capping

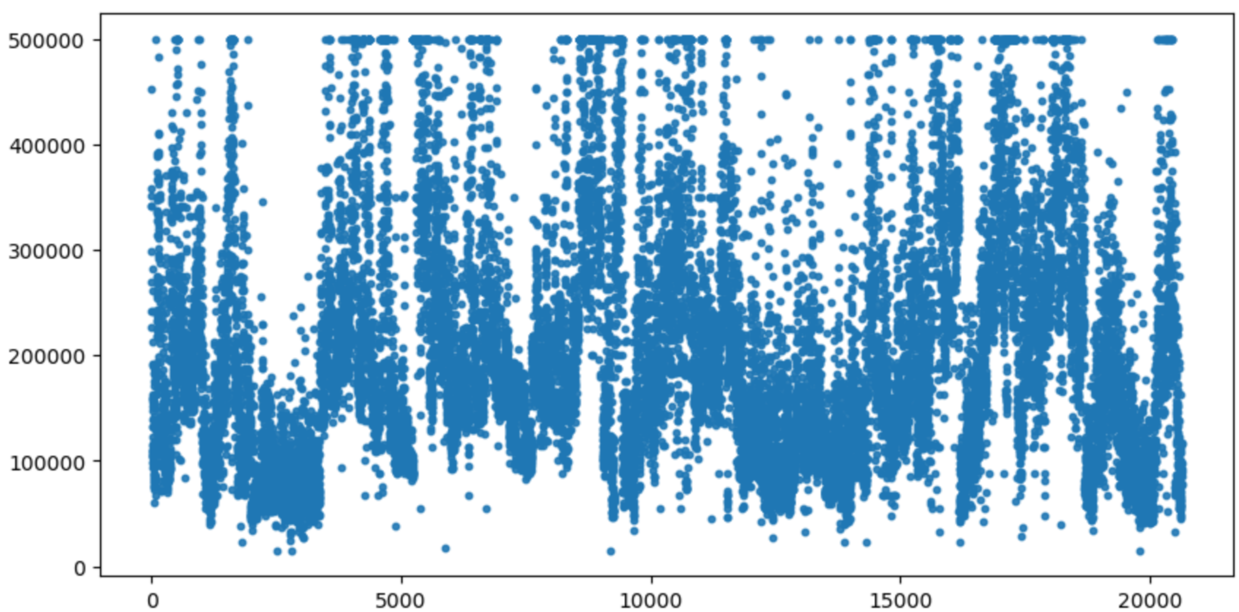

Figure 1: Scatterplot showing capped median house values

2.3 Outlier Removal

Boxplots helped detect and remove outliers:

- `total_rooms`: 1246
- `total_bedrooms`: 492
- `population`: 448
- `median_income`: 310

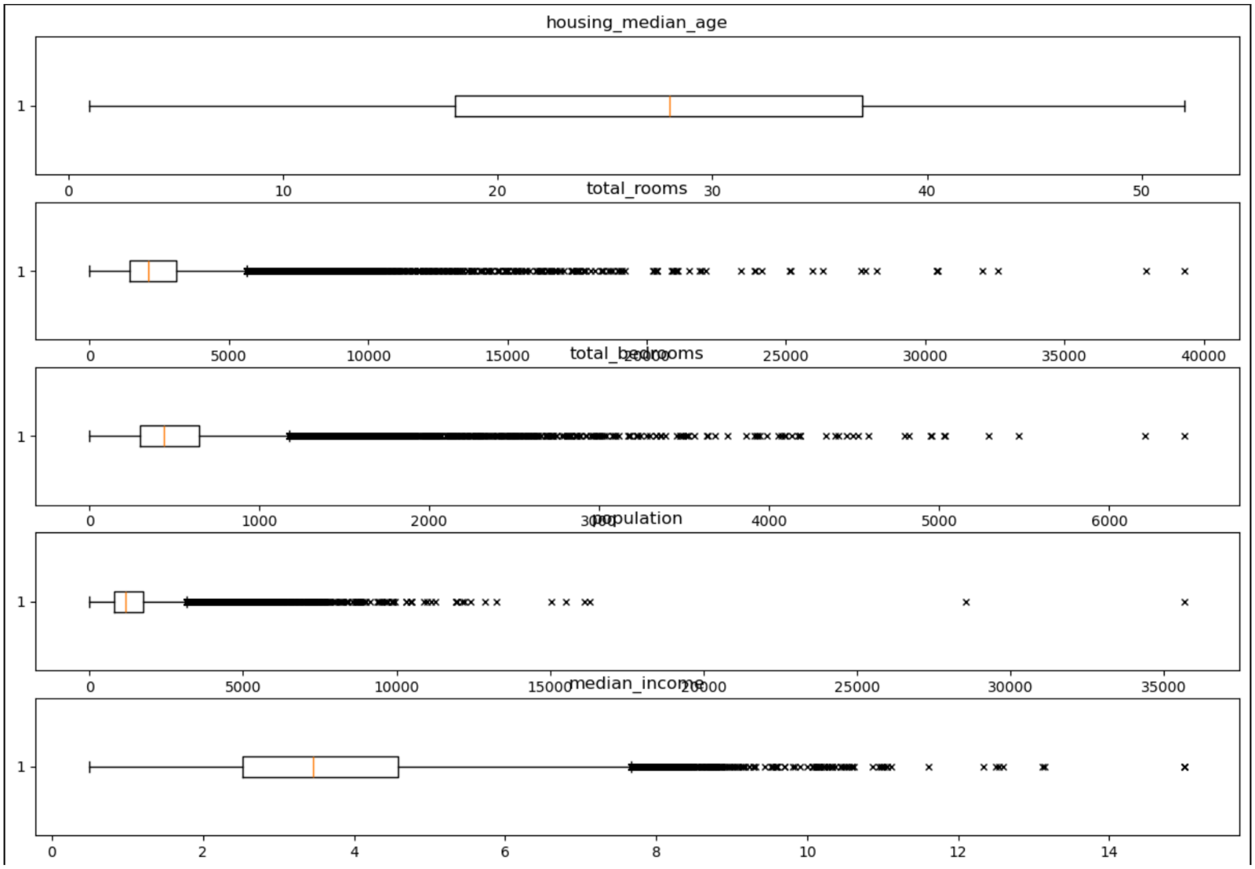

Figure 2: Boxplot showing feature outliers

2.4 Feature Selection and Scaling

`total_bedrooms` was dropped due to high multicollinearity and low correlation with the target. Moreover dropping it procued a lower overall cost than some other modifications. Data was standardized using `StandardScaler` and split into 80% training and 20% testing sets.

3 Pure Python Implementation

Implemented batch gradient descent using only fundamental Python structures and loops.

3.1 Results

- Time: **358.24 seconds**
- Bias term b: `187301.60`
- Weight vector $\mathbf{w}$:

```
[-68066.41, -71692.31, 10230.93, -3905.72, -42967.43,
```

```
49503.31, 52540.76, 15727.69, 3189.96, 5094.10, 5411.11]
```

Metric	Value
MAE	44106.24
RMSE	59178.57
R^2 Score	0.60

Table 1: Pure Python Model Performance

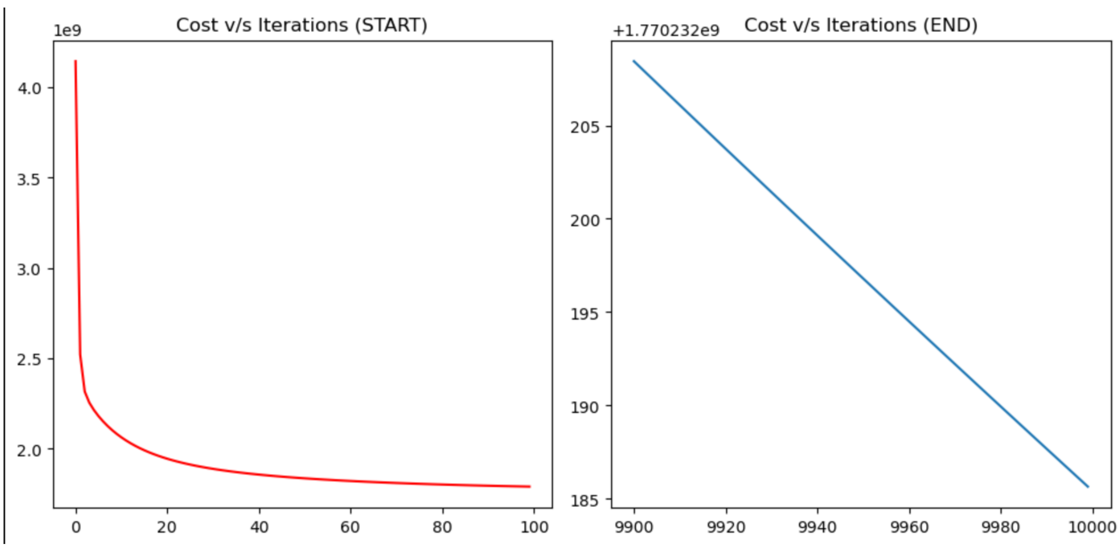

Figure 3: Cost vs. Iterations for Python Implementation

4 NumPy Implementation

4.1 Overview

We reimplemented gradient descent using NumPy arrays, allowing efficient vectorized computation.

4.2 Results

- Time: **15.89 seconds**
- Bias term b: `187301.60`
- Weight vector $\mathbf{w}$:

```
[-68066.41, -71692.31, 10230.93, -3905.72, -42967.43,
 49503.31, 52540.76, 15727.69, 3189.96, 5094.10, 5411.11]
```

Metric	Value
MAE	44106.24
RMSE	59178.57
R^2 Score	0.60

Table 2: NumPy Model Performance

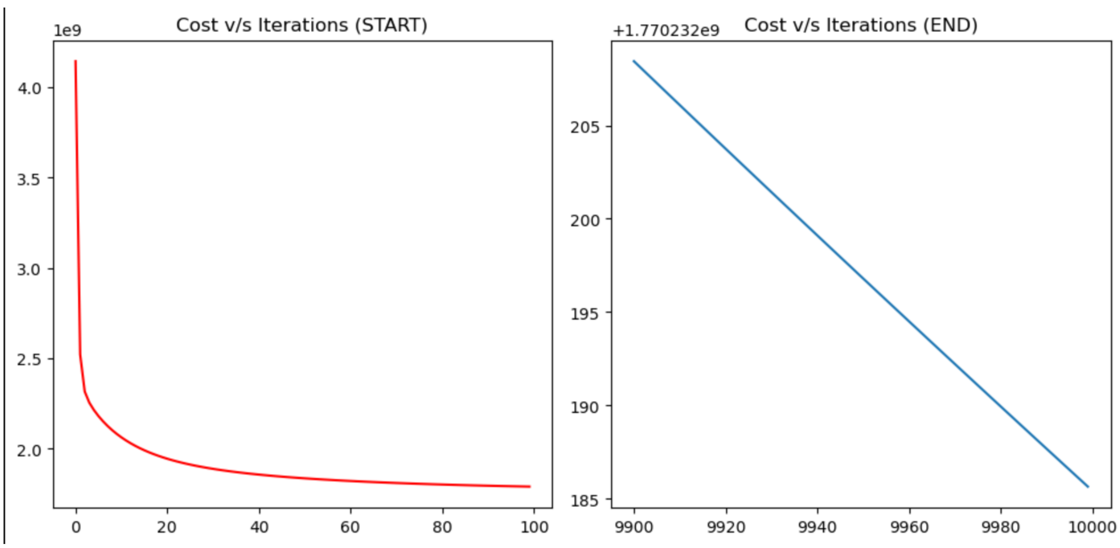

Figure 4: Cost vs. Iterations for NumPy Implementation

5 Scikit-learn Implementation

Used `LinearRegression` for exact solution via SVD(Singular Value Decomposition).

- Time: **0.01 seconds**
- Bias term b: `187301.60`
- Weight vector $\mathbf{w}$:

```
[-68066.99, -71693.48, 10230.54, -3907.58, -42968.58,
49506.20, 52540.47, 16761.64, 4174.78, 5744.91, 6106.49]
```

Metric	Value
MAE	44105.74
RMSE	59177.91
R^2 Score	0.60

Table 3: Scikit-learn Model Performance

6 Analysis

6.1 Performance Summary

All models roughly achieved:

- MAE: 44106
- RMSE: 59178
- R^2: 0.60

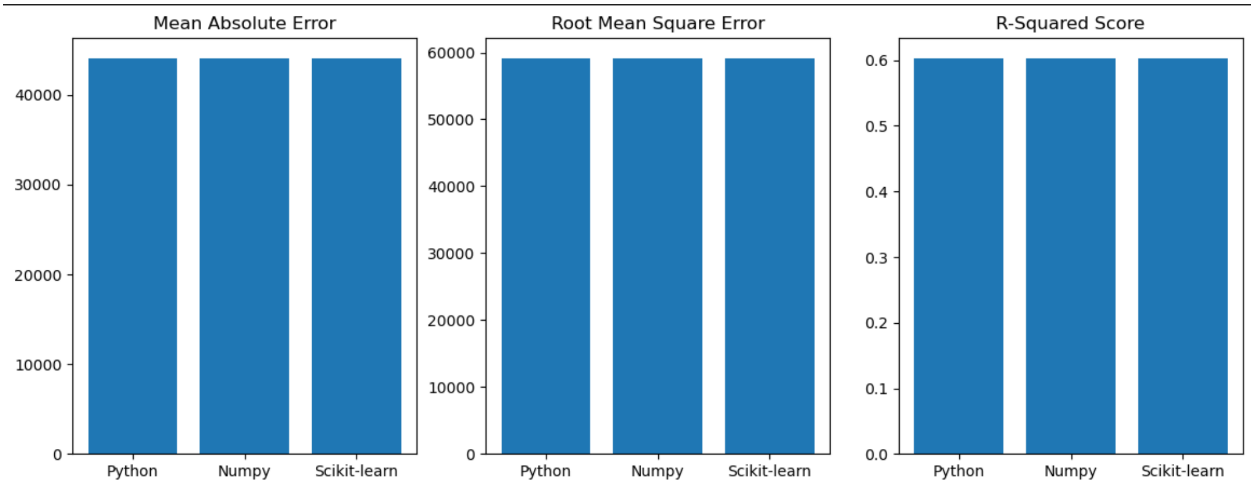

Figure 5: MAE, RMSE, R^2 Comparison

6.2 Training Time Insight

The difference in training times reflects the computational efficiency of each approach:

- Pure Python: slowest due to nested loops
- NumPy: faster due to vectorized matrix operations as it allows parallel processing of data using modern CPU's SIMD capabilities
- Scikit-learn: fastest via analytical closed-form solution

6.3 Interpreting Metrics

- MAE $\sim$ \$44K is acceptable but high — can be improved with better features
- RMSE reveals that some predictions deviate substantially
- $R^2 = 0.60$ means the model explains 60% of price variation

6.4 About Iterations and Learning Rate

10,000 iterations ensured convergence in custom implementations. Learning rate α=0.7 was chosen with care:

- Too large: cost overshoots and diverges
- Too small: slow convergence

Empirical tuning was used to achieve a stable, steadily decreasing cost curve.

Conclusion

This project explored multivariable linear regression through three different implementations. While all yielded identical results, their computational characteristics varied:

- Pure Python: Educational but slow.
- NumPy: Efficient and scalable.
- Scikit-learn: Fastest and production-ready.

The choice of implementation depends on the context: clarity and learning for beginners, vectorization for practical use, and libraries like Scikit-learn for professional deployment.